Universidad Nacional Mayor de San Marcos

Universidad del Perú, Decana de América

Facultad de Ingeniería Electrónica y Eléctrica

Escuela Profesional de Ingeniería Electrónica

Diseño e implementación de un filtro gaussiano usando FPGA

Autor

Zambrano Rodríguez Jorge Armando

Lima, Perú

2021

TABLA DE CONTENIDO

Índice de figuras

Indice de tablas

Capítulo 1 INTRODUCCIÓN

1.1 Planteamiento del problema

La importancia de los filtros electrónicos se debe a su gran diversidad de aplicaciones, como por ejemplo, ecualizadores de audio, cancelación de ruido de una señal, procesamiento de señales biomédicas, voz, imágenes, telecomunicaciones, audio digital, ingeniería sísmica, entre otros.

Los filtros digitales, en general, son implementados en procesadores digital de señales (DSP) debido a su arquitectura definida y su soporte en software ya establecida. Sin embargo, un FPGA también presenta grandes ventajas ante la necesidad de procesar señales digitales pues permite la implementación de sistemas tan complejos como microprocesadores o sistemas en un solo circuito integrado además de que el costo de desarrollo es relativamente bajo. Así mismo, se hace uso de software especializados como Quartus de Altera o ISE de Xilinx. Los FPGAs están sustituyendo a los arreglos de compuertas y a algunos ASIC debido a sus ventajas y gran potencial de desarrollo tecnológico, además, da la posibilidad de integrar partes digitales y analógicas; por otro lado, cuentan con mayores frecuencias de operación que los DSP y pueden reprogramarse con facilidad.

Debido a su gran demanda actual, se hace indispensable el estudio y comprensión del funcionamiento del FPGA. Por tanto, el presente trabajo tiene como objetivo realizar la implementación de un filtro digital gaussiano en un dispositivo FPGA para reducir el ancho de banda de los pulsos digitales que se desean transmitir en un canal de comunicación con ancho de banda limitado. Para ello, se requiere el diseño teórico del filtro, la descripción modular del circuito, la síntesis del código y la programación del FPGA. Finalmente, se verifica el correcto funcionamiento del sistema mediante la interacción con la placa de desarrollo, ingresando valores que representaran señales de impulso, escalón y sinusoides.

1.2 Objetivos

1.4.1 Objetivo general

Realizar el diseño e implementación un filtro digital gaussiano en un FPGA para experimento de laboratorio en tiempo real.

1.4.2 Objetivo específicos

Diseñar un bloque digital de un filtro FIR.

Diseñar un filtro gaussiano a partir de filtros FIR fundamentales.

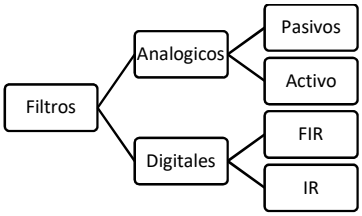
Diseñar un filtro polifásico a partir de filtro FIR fundamentales.

Capítulo 2 MARCO TEÓRICO

2.1 Filtro

Un filtro es un dispositivo (bien realizado en hardware o software) que se aplica a un conjunto de datos ruidosos para poder extraer información sobre una cantidad de interés. En el área de las señales, el filtrado es un proceso mediante el cual se modifica el contenido espectral de una señal.
El término filtro se utiliza comúnmente para describir un dispositivo que discrimina aquello que pasa a través de él. Así, por ejemplo, un filtro de aire permite que sólo pase aire a través de éste, evitando que las partículas de polvo presentes en el aire lo atraviesen.

"When you think about it, everything is a filter" (Julius Smith).



2.1.1 Filtro digital

Es un filtro que opera sobre señales digitales. Es una operación matemática que toma una secuencia de números (la señal de entrada) y la modifica produciendo otra secuencia de números (la señal de salida) con el objetivo de resaltar o atenuar ciertas características. Puede existir como una fórmula en un papel, un loop en un programa de computadora, como un circuito integrado en un chip.

El filtro digital es un sistema lineal e invariante en el tiempo (LTI) que modifica el espectro en frecuencia de la señal de entrada X(w), según la respuesta que tenga en frecuencia H(w) (más conocida como función de transferencia), para dar lugar a una señal de salida con espectro Y(w) = H(w) * X(w). En cierto sentido, H(w) actúa como una función de ponderación o función de conformación espectral para las diferentes componentes frecuenciales de la señal de entrada.

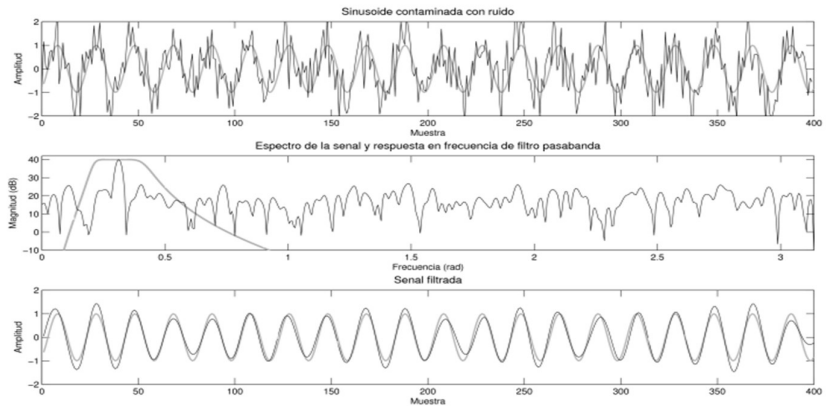


Figura 1. Proceso de filtrado

2.2 Filtro FIR

Un filtro FIR es aquel que tiene una respuesta finita al impulso y que se caracterizan por ser sistemas no recursivos.

En este tipo de filtrado no existe retroalimentación. Además, la respuesta al impulso H(w), es de duración finita ya que si la entrada se mantiene en cero durante L periodos consecutivos la salida también será cero. Algunas de las características de este tipo de filtros son las siguientes:

- Un filtro FIR puede ser diseñado para tener fase lineal.
- Siempre son estables porque son hechos únicamente con ceros en el plano complejo.
- Los errores por desbordamiento no son problemáticos porque la suma de productos en un filtro FIR es desempeñada por un conjunto finito de datos.
- La salida siempre es una combinación lineal de los valores presentes y pasados de la señal de entrada.
- Tiene memoria finita

2.2.1 Ecuaciones en diferencias

Un filtro FIR de orden N se describe mediante la ecuación en diferencias:

$$y(n) = b_0 x(n) + b_1 x(n-1) + \ldots + b_{N-1} x(n-N+1) = \sum_{k=0}^{N-1} b_k x(n-k)$$

Donde la secuencia b_k son los coeficientes del filtro.

La salida {y(n)} puede escribirse como la convolución de la entrada {x(n)} con la respuesta impulsional {h(n)}:

$$y(n) = \sum_{k=-\infty}^{+\infty} h(k) \cdot x(n-k) \qquad con \quad h(k) = \begin{cases} 0 & k < 0 \\ h(k) & 0 \le k < N \\ 0 & k \ge N \end{cases}$$

Luego la expresión puede reescribirse como:

$$y(n) = \sum_{k=0}^{N-1} h(k) \cdot x(n-k)$$

Donde bk={h(k)}.

A partir de esta ecuación en diferencias puede obtenerse la función de transferencia del filtro en el dominio de Z:

$$H(z) = \sum_{K=0}^{N-1} h[k].\, z^{-k}$$

2.2.2 Modelo diagrama de bloques

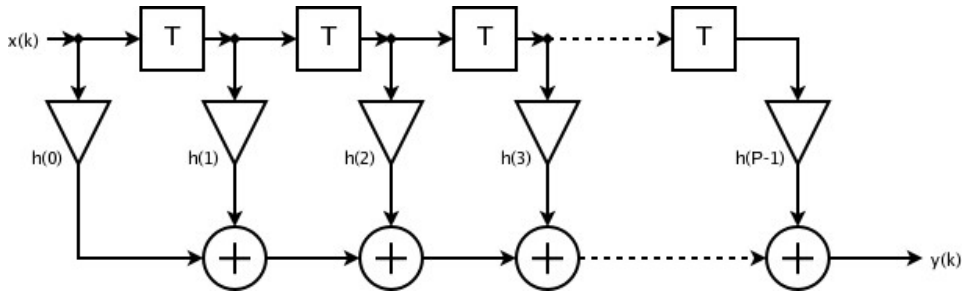


Figura 2. Diagrama de bloques de un filtro FIR

La interpretación gráfica de un filtro LTI FIR de orden L se realiza directamente a partir de su ecuación en diferencias, tal y como se muestra en la Figura . Ésta es la denominada Implementación con Estructura Directa. Puede observarse que para su realización se utilizan básicamente una línea de elementos de retardo, multiplicadores y sumadores, con un número de etapas igual al orden del filtro.

2.2.3 Diseño de un filtro bajo el método de las ventanas

FUNCIÓN VENTANA

Las ventanas son funciones matemáticas usadas con frecuencia en el análisis y el procesamiento de señales para evitar las discontinuidades al principio y al final de los bloques analizados.

En el procesamiento de señales, una ventana se utiliza cuando el análisis se centra en una señal de longitud voluntariamente limitada. En efecto, una señal real tiene que ser de tiempo finito; además, un cálculo sólo es posible a partir de un número finito de puntos. Para observar una señal en un tiempo finito, se multiplica por una función ventana.

El objetivo de usar ventanas, que no contengan discontinuidades abruptas en sus características en el dominio en el tiempo, es la de reducir los efectos de rizado no deseados en la respuesta en frecuencia del filtro.

Nombre de la ventana	Secuencia en el dominio del tiempo. $h(n), 0 \le n \le M-1$
Bartlett (triangular)	$1 - \frac{2\left\lvert n - \frac{M-1}{2} \right\rvert}{M-1}$
Blackman	$0.42 - 0.5\cos\frac{2\pi n}{M-1} + 0.08\cos\frac{4\pi n}{M-1}$
Hamming	$0.54 - 0.46\cos\frac{2\pi n}{M-1}$
Hanning	$\frac{1}{2}\left(1 - \cos\frac{2\pi n}{M-1}\right)$
Kaiser	$\frac{I_0\left[\alpha\sqrt{\left(\frac{M-1}{2}\right)^2 - \left(n - \frac{M-1}{2}\right)^2}\right]}{I_0\left[\alpha\left(\frac{M-1}{2}\right)\right]}$

Figura 3. Tipos de función ventana

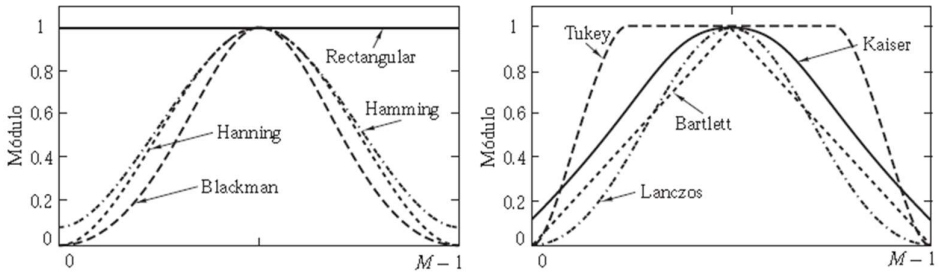


Figura 4. Grafico de comparación de ventanas

FILTRO FASE LINEAL

La ventaja de los filtros FIR es que pueden diseñarse para que presenten FASE LINEAL. La linealidad de fase implica que se verifiquen ciertas condiciones de simetría:

- Un sistema no causal con respuesta impulsional conjugada simétrica $(h(n) = h^*(-n))$ tiene una F. de Transferencia real.
- Un sistema no causal con respuesta impulsional conjugada $(h(n) = -h^*(-n))$ tiene una Función de transferencia imaginaria pura.

Es decir tendremos fases que pueden cero 0 ó 2 π, si queremos que las secuencias sean realizables, las retardaremos un número de muestras adecuado para que se transformen en causales.

Si consideramos sistemas FIR con coeficientes reales, una secuencia conjugada simétrica se dice que es una secuencia PAR, y una secuencia conjugada anti simétrica es una secuencia IMPAR. Dependiendo del número de coeficientes del filtro y del tipo de simetría tenemos varias posibilidades.

Tipo:	Número de términos (N)	Simetría
I	Impar	Simétrico $h(k) = h(N-1-k)$
II	Par	Simétrico $h(k) = h(N-1-k)$
III	Impar	Antisimétrico $h(k) = -h(N-1-k)$
IV	Par	Antisimétrico $h(k) = -h(N-1-k)$

Figura 5. Tipos de simetria

UBICACIÓN DE POLOS Y CEROS:

Los filtros FIR solo presentan polos en el origen, por lo que siempre son estables. Sobre el posicionamiento de los ceros, resulta fácil demostrar que en los filtros de fase lineal los ceros se dan en pares recíprocos, es decir, si z_0 es una raíz del polinomio H(z), también lo será $z_0{}^{-1}$. Veámoslo:

La función de transferencia de un filtro FIR es:

$$H(z) = \sum_{K=0}^{N-1} h[k].z^{-k}$$

Si sustituimos z por z^{-1},

$$H(z^{-1}) = \sum_{k=0}^{N-1} h(k)\cdot z^{k} = z^{N-1}\cdot \sum_{k=0}^{N-1} h(k)\cdot z^{-N+1+k}$$

Realizando un cambio de variable en el índice del sumatorio de forma que k'=N-1-k,

$$H(z^{-1}) = z^{N-1}\cdot \sum_{k'=0}^{N-1} h(N-1-k')\cdot z^{-k'}$$

Si además se trata de un FIR de fase lineal, h(k) = ±h(N −1− k) , de lo que derivamos que las raíces del polinomio H(z -1) son también raíces de H(z).

Si además queremos que los coeficientes sean reales, las raíces complejas deben aparecer en forma de pares complejo conjugados.

MÉTODO DE LAS VENTANAS

El método de las ventanas se basa en acotar la respuesta impulsional infinita de un filtro Ideal. Si queremos implementar un filtro pasa baja con una respuesta ideal (transición abrupta de la banda pasante a la atenuada), la respuesta impulsional es infinita y no causal. Para obtener un filtro FIR realizable se puede proponer truncar h(n) y retardarla hasta convertirla en causal.

La respuesta en frecuencia de un del filtro ideal con características de pasa bajo viene dada por:

$$H_d(\Omega) = \begin{cases} e^{j\Omega\beta} & |\Omega| \leqslant \Omega_c \\ 0 & \Omega_c < |\Omega| \leqslant \pi \end{cases}$$

Dónde: β= pendiente de la fase lineal
Ω_c=frecuencia de corte

La respuesta al impulso del filtro ideal será:

$$h_d[n] = \frac{sen[\Omega_c(n-\beta)]}{\pi(n-\beta)}$$

Tenemos en cuenta que h_d infinita y no causal.

En esta parte se va a truncar h_d por una función de ventana w[n] de las que revisamos anteriormente.

$$h[n] = h_d[n].w[n]$$

$$h[n] = \frac{sen[\Omega_c(n-\beta)]}{\pi(n-\beta)}.w[n]$$

Haciendo este truncamiento hacemos que el filtro sea finito.

Luego para que la fase sea lineal hacemos $\beta=\frac{N-1}{2}$, esto nos da una simetría alrededor de β. También hace que el filtro sea causal.

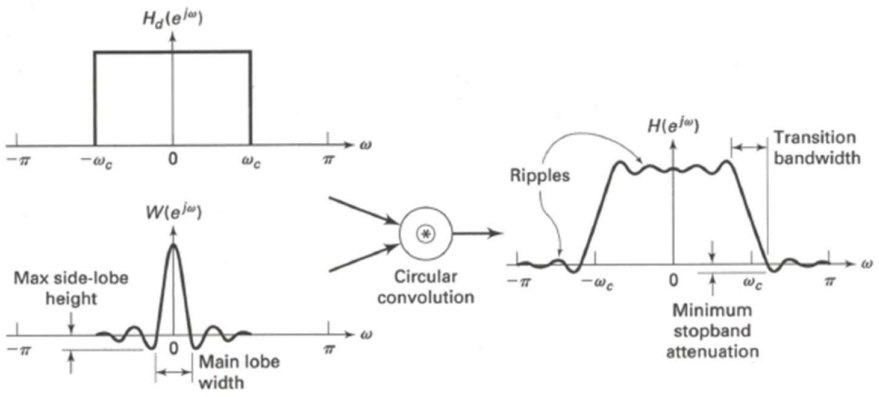


Figura 6. Método de ventana

Finalmente se obtiene el filtro FIR:

$$h[n] = \frac{sen[\Omega_c(n-(\frac{N-1}{2}))]}{\pi(n-(\frac{N-1}{2}))}.w[n] \quad ; \quad 0 \leqslant n \leqslant N-1$$

Tipo de ventana	Anchura aproximada de la región de transición del lóbulo principal	Pico del lóbulo secundario (dB)
Rectangular	$4\pi/M$	−13
Bartlett	$8\pi/M$	−25
Hanning	$8\pi/M$	−31
Hamming	$8\pi/M$	−41
Blackman	$12\pi/M$	−57

Figura 7. Características de las ventanas

Para los coeficientes del filtro fir, estos se van a obtener del producto de h_d y w.

h[0]; h[1]; h[2]; h[3]; ….; h[N-1]

asi:

h[n]=h[0].δ[n]+ h[1].δ[n-1]+ h[2].δ[n-2]+ h[3].δ[n-3] + … + h[N-4].δ[n-(N-1)]

Realizando la transformada Z:

$$H(z) = \sum_{k=0}^{N-1} h[k].z^{-1}$$

Desarrollando la ecuación se puede observar que los polos se encuentran en el origen para un filtro FIR.

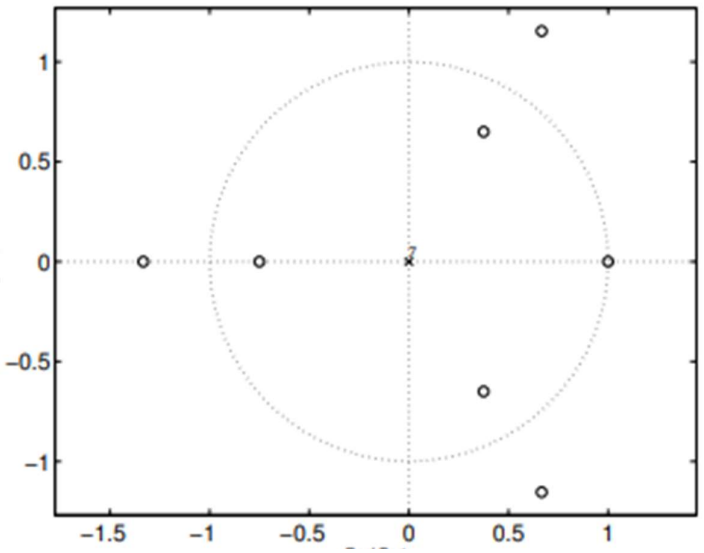


Figura 8. Distribución de polos y ceros

Ejemplo práctico:

Haciendo uso de un software online ingresamos los parámetros para un filtro pasabajo:

Fm=1khz (frecuencia de muestreo)

Fc=300 (frecuencia de corte)

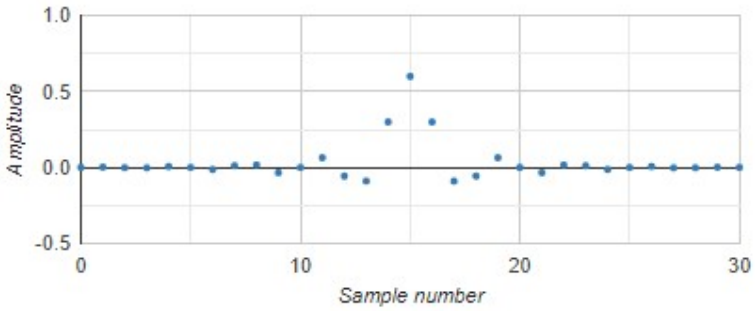
BL=100 (banda de transición)

Ventana: Hamming

-Resultados:

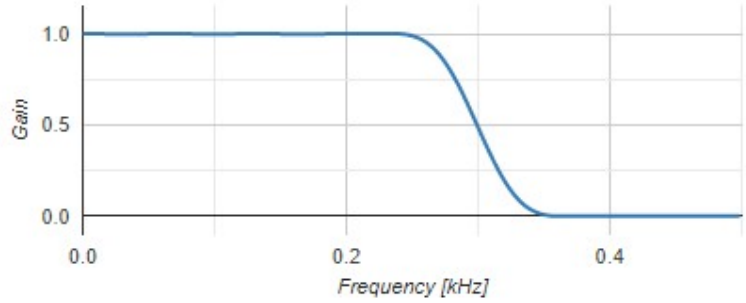
Un filtro con 31 coeficientes.

Respuesta al impulso:



Aquí podemos observar los 31 coeficientes del filtro

Respuesta en frecuencia:



Como se observa en el grafico se ha atenuado el rizado que aparece cuando no se usa una ventana haming u otra. También se observa como este filtro va a dejar pasar las frecuencias hasta 300Hz aproximadamente.

2.3 Filtro digital Gaussiano

Un filtro gaussiano es un filtro cuya respuesta al impulso es una función gaussiana (o una aproximación a ella, ya que una verdadera respuesta gaussiana es físicamente irrealizable ya que tiene soporte infinito). Los filtros gaussianos tienen la propiedad de no sobrepasar una entrada de función escalonada y minimizar el tiempo de subida y bajada. Este comportamiento está estrechamente relacionado con el hecho de que el filtro gaussiano tiene el retardo de grupo mínimo posible.

El objetivo de esta parte será hacer un filtro adecuado para hacerlo convolucionar una entrada impulso unitario δ(t) , ya que este impulso unitario en el dominio de la frecuencia tiene un ancho de banda infinito.

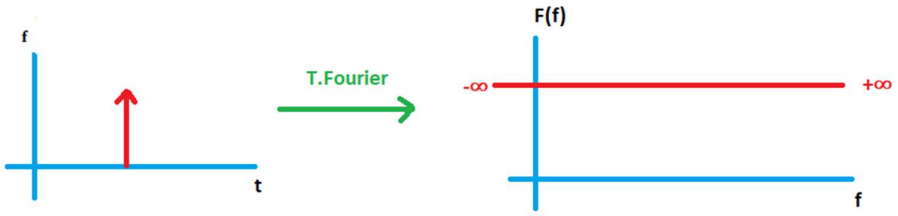


Figura 9. Transformada de Fourier de un pulso unitario

Entonces lo que se hace es de hacer convolucionar por un filtro gaussiano:

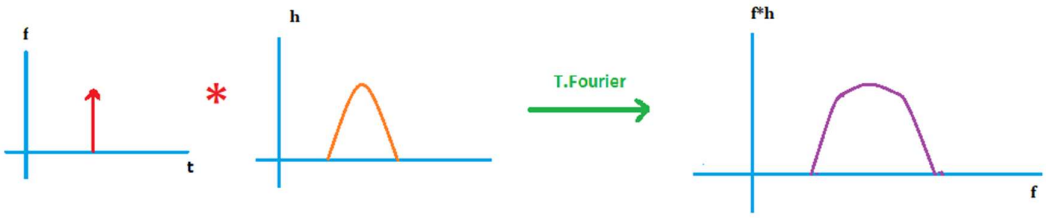


Figura 10. Transformada de Fourier de la convolución
de un pulso y el filtro gaussiano.

Y como resultado el ancho de banda del impulso se reduce y es posible enviarlo a un modulador para su procesamiento.

La aplicación de este tipo de filtrado se puede encontrar en la MODULACION POR DESPLAZAMIENTO DE FRECUENCIA GAUSSIANA En lugar de modular directamente la frecuencia con los símbolos de datos digitales, cambiando "instantáneamente" la frecuencia al comienzo de cada período de símbolo, la codificación por desplazamiento de frecuencia gaussiana (GFSK) filtra los pulsos de datos con un filtro gaussiano para suavizar las transiciones. Este filtro tiene la ventaja de reducir la potencia de banda lateral, reduciendo la interferencia con los canales vecinos, a costa de incrementar la interferencia entre símbolos

2.2.1 Modelo matemático

El diseño de filtro FIR con forma de pulso gaussiano se realiza truncando una versión muestreada de la respuesta al impulso en tiempo continuo del filtro gaussiano que viene dada por:

$$h(t) = \frac{\sqrt{\pi}}{a} e^{-\frac{\pi^2.t^2}{a^2}}$$

El parámetro 'a' está relacionado con el producto ancho de banda de 3 dB con el tiempo de duración del símbolo (B * Ts) del filtro gaussiano dado por:

$$a = \frac{1}{B.T_s}\sqrt{\frac{\log(2)}{2}}$$

Símbolo = Son las unidades de señal que forman parte del dominio discreto.

Ts = Tiempo de duración de cada símbolo, en segundo.

B= ancho de banda de 3 dB, en hercios.

bt= B.Ts = Producto del ancho de banda de 3 dB y el tiempo del símbolo.

Hay dos errores de aproximación en este diseño: un error de truncamiento y un error de muestreo. El error de truncamiento se debe a una aproximación de tiempo finito (FIR) de la respuesta al impulso teóricamente infinita del filtro gaussiano ideal. El error de muestreo (aliasing) se debe al hecho de que una respuesta de frecuencia gaussiana no es realmente de bandas limitada en un sentido estricto (es decir, la energía de la señal gaussiana más allá de una cierta frecuencia no es exactamente cero). Esto se puede observar en la función de transferencia del filtro gaussiano de tiempo continuo, que se muestra a continuación:

$$H(f) = e^{-a^2 f^2}$$

A medida que f aumenta, la respuesta en frecuencia tiende a cero, pero nunca es exactamente cero, lo que significa que no se puede muestrear sin que se produzca algún aliasing.

$$h(t) = \frac{\sqrt{\pi}}{a} e^{-\frac{\pi^2.t^2}{a^2}} \quad \text{---------TF---------} > \quad H(f) = e^{-a^2 f^2}$$

2.2.2 Diseño de un filtro gaussiano con el software Matlab

Para diseño del filtro FIR Gaussiano utilizando la función gaussdesign. Las entradas a esta función son el producto de tiempo de símbolo y el ancho de banda de 3 dB, el número de períodos de símbolo entre el inicio y el final de la respuesta al impulso del filtro, es decir, el intervalo del filtro en símbolos, y el factor de sobre muestreo “sps” (es decir, el número de muestras por símbolo).

El factor de sobre muestreo determina la frecuencia de muestreo y la longitud del filtro y, por lo tanto, juega un papel importante en el diseño del filtro FIR de Gauss. Los errores de aproximación en el diseño se pueden reducir con una elección adecuada del factor de sobre muestreo.

La función h = gaussdesign(bt,span,sps) diseña un filtro de forma de pulso FIR gaussiano de paso bajo y nos devuelve un vector h[] de coeficientes del filtro h. El filtro se trunca en

símbolos y cada período de símbolo contiene muestras.spansps El orden del filtro, , debe ser par sps*span.

Parámetros de entrada:

bt — Producto de tiempo de símbolo de ancho de banda de 3 dB

Producto del ancho de banda de 3 dB, en hercios, y el tiempo del símbolo, en segundos. Especifique este valor como un escalar real positivo. Los valores más pequeños de producir anchos de pulso más grandes

span — Número de símbolos

Número de símbolos, especificado como un escalar entero positivo.

sps — Muestras por símbolo

Número de muestras por período de símbolo (factor de sobremuestreo), especificado como un escalar entero positivo.

Parametro de salida

h — Coeficientes de filtro FIR

Coeficientes FIR del filtro gaussiano de forma de pulso, son devueltos como vector de fila.

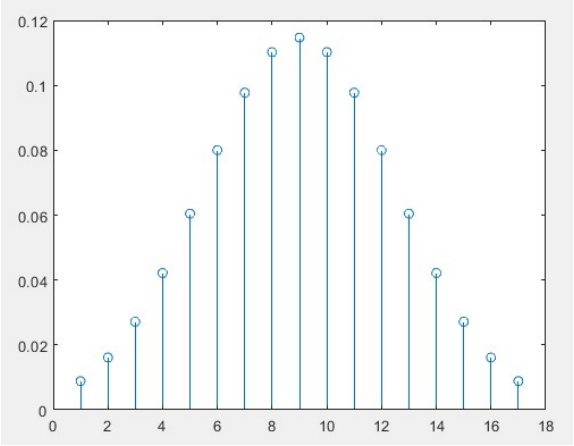
Ejemplo practico:

Usando el software Matlab ingresamos los siguientes parámetros de la función gaussdesing:

Bt= 0.3 , span=2 , sps=8

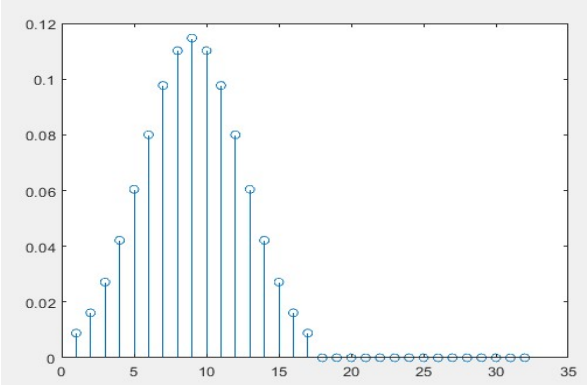
-Resultados

Un filtro de 17 coficientes.



Se muestra el filtro de gauss con sus coeficientes.

Respuesta al impulso:



Se puede apreciar como el ancho de banda del pulso es reducido.

2.3 Filtro Polifásico

Debido a la naturaleza de los procesos de diezmado e interpolación, se pueden desarrollar estructuras de filtros polifásicos para implementar de manera eficiente los filtros de diezmado e interpolación (utilizando un menor número de multiplicaciones y adiciones). Como explicaremos, estos filtros son filtros de paso total con diferentes cambios de fase (Proakis y Manolakis, 2007), por eso los llamamos *filtros polifásicos.*

Suponemos que el filtro de interpolación FIR tiene cuatro taps, que se muestran como

$$H(z)= h(0)+h(1)z^{-1} + h(2)z^{-2} + h(3)z^{-3}$$

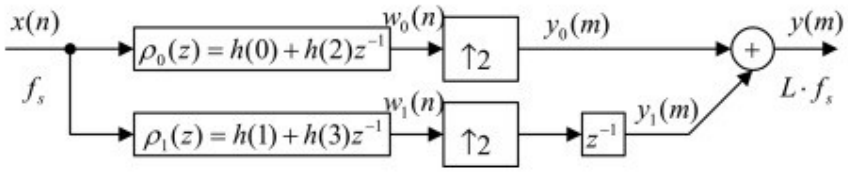
y la salida del filtro es

$$y(m)= h(0)w(m)+h(1)w(m-1)+h(2)w(m-2)+h(3)w(m-3$$

Para fines de comparación, el proceso de interpolación directa que se muestra en la figura 11.17 se resume en la siguiente tabla, donde *w* (*m*) es la señal muestreada *ey* (*m*) la salida interpolada. El procesamiento de cada muestra de entrada *x* (*n*) requiere aplicar la ecuación de diferencia dos veces para obtener *y* (0) e *y* (1). Por lo tanto, para este ejemplo, necesitamos ocho multiplicaciones y seis sumas.

norte	*x* (*n*)	*metro*	*w* (*m*)	*y* (*m*)
n = 0	*x* (0)	*m* = 0	*w* (0) = *x* (0)	*y* (0) = *h* (0) *x* (0)
		m = 1	*w* (1) = 0	*y* (1) = *h* (1) *x* (0)
n = 1	*X* (1)	*m* = 2	*w* (2) = *x* (1)	*y* (2) = *h* (0) *x* (1) + *h* (2) *x* (0)
		m = 3	*w* (3) = 0	*y* (3) = *h* (1) *x* (1) + *h* (3) *x* (0)
n = 2	*x* (2)	*m* = 4	*w* (4) = *x* (2)	*y* (4) = *h* (0) *x* (2) + *h* (2) *x* (1)
		m = 5	*w* (5) = 0	*y* (5) = *h* (1) *x* (2) + *h* (3) *x* (1)
...	...	...	...	...

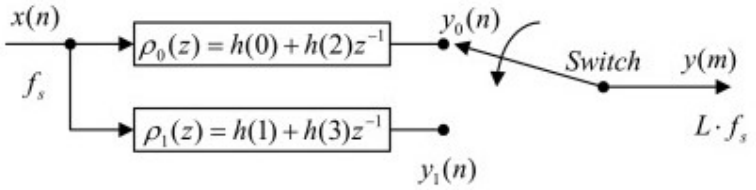
Los resultados de salida de la tabla presentada se pueden obtener fácilmente utilizando los filtros polifásicos que se muestran en el siguiente diagrama .



En general, existen filtros polifásicos *L.* Teniendo el filtro de interpolación diseñado *H* (*z*) de *N* derivaciones, podemos determinar cada banco de coeficientes de filtro siguiente manera:

$$\rho_k(n) = h(k + nL) \text{ for } k = 0, 1, \ldots, L - 1$$

Por lo tanto, el primer filtro ρ_0 (*z*) tiene los coeficientes *h* (0) y *h* (2). De manera similar, el segundo filtro ρ_1 (*z*) tiene coeficientes *h* (1) y *h* (3). De hecho, los coeficientes de filtro de ρ_0 (*z*) son una versión diezmada de 1, y así sucesivamente.*h* (*n*) comenzando en $k = 0$, mientras que los coeficientes de filtro de ρ_1 (*z*) son una versión diezmada de *h* (*n*)



2.4 Sistemas digitales

2.4.1 Matrices de Puertas Programable por Campo (FPGA)

Un FPGA consiste en un arreglo de bloques lógicos programables, bloques de entrada/salida (E/S) y una red de interconexiones programable que puede ser usada para conectar un elemento lógico a cualquier otro. Cada bloque lógico contiene la circuitería necesaria para implementar lógica combinacional o secuencial arbitraria para realizar funciones deseadas, la flexibilidad de interconexión de estos bloques es la razón las que se consiguen diseños de gran complejidad y con arquitecturas que brindan gran rendimiento haciendo el FPGA óptimo para las tareas de procesamiento.

2.4.2 VHDL

Se define VHDL como el lenguaje de descripción de hardware, la cual describe el comportamiento de un circuito electrónico o sistema, del cual se puede implementar un circuito físico. Sus siglas significan Lenguaje de descripción de Hardware. Fue el primer, y original, lenguaje de descripción de hardware en ser estandarizado por el Instituto de Ingenieros Eléctricos y Electrónicos (IEEE) en estándar IEEE 1076.

VHDL está diseñado para sintetizar y simular circuitos, sin embargo, aunque es completamente simulable, no todos los circuitos son sintetizables.

VHDL proporciona tres métodos básicos para describir un circuito digital por software: algorítmico, flujo de datos y estructural.

2.4.3 Estilo estructural de programación en VHDL

El método estructural para escribir un descripción VHDL de una función lógica puede compararse con el montaje de circuitos integrados en una tarjeta de circuito y el establecimiento de las interconexiones entre ellos mediante cables. Mediante el método estructural, se describen las funciones lógicas y se especifica cómo se conectan entre sí. El componente VHDL es una forma de predefinir una función lógica para poder emplearla repetidas veces en un mismo programa o en otros programas. El componente puede utilizarse para describir cualquier cosa, desde una simple puerta lógica a una función lógica compleja. La señal VHDL es una forma de especificar una conexión mediante un "cable" entre componentes.

2.4.4. Máquina de estado finito (FSM)

Es un circuito que contiene un número predeterminado de estados, en el cual la máquina solo puede existir en un estado a la vez. El circuito transita entre estados a partir de un evento de disparo, comúnmente un flanco de reloj, y los valores de las entradas de la máquina; todas las posibles transiciones también están predeterminados. A través del uso de estados y las secuencias de transición pasadas, el circuito es capaz de decidir su siguiente estado, esto le permite crear salidas que son más inteligentes comparadas a un circuito de lógica combinacional simple que solo tiene salidas basadas en el valor actual de sus entradas.

Hay dos tipos diferentes de condiciones de salida para una máquina de estado. El primero es cuando la salida solo depende del estado actual de la máquina, este tipo de sistema es llamado Máquina de Moore. El segundo es cuando las salidas dependen del estado actual y de las entradas al sistema, se lo conoce como Máquina de Mealy.

Capítulo 3 METODOLOGÍA DE DESARROLLO DEL PROYECTO

3.1 Tecnología embebida a emplearse en el desarrollo del proyecto

3.1.3 Software Quartus II 12.0 Altera

Es una herramienta de software producida por Altera para el análisis y síntesis de diseños realizados en HDL. Este software permite al desarrollador compilar sus diseños, realizar análisis temporales, examinar diagramas RTL y configurar el dispositivo de destino con el programador.

La edición web es una versión gratuita de Quartus II que puede ser descargada o enviada gratuitamente por correo. Esta edición permite la compilación y la programación de un número limitado de dispositivos altera.

3.1.1 Placa de desarrollo Cyclone IV

Para la parte experimental se optó por utilizar la placa de desarrollo basada en un Cyclone IV EP4CE6E22C8N que posee diferentes periféricos que facilitarán el ingreso de datos y la visualización de las salidas. El reloj principal de este módulo es de 50MHz.

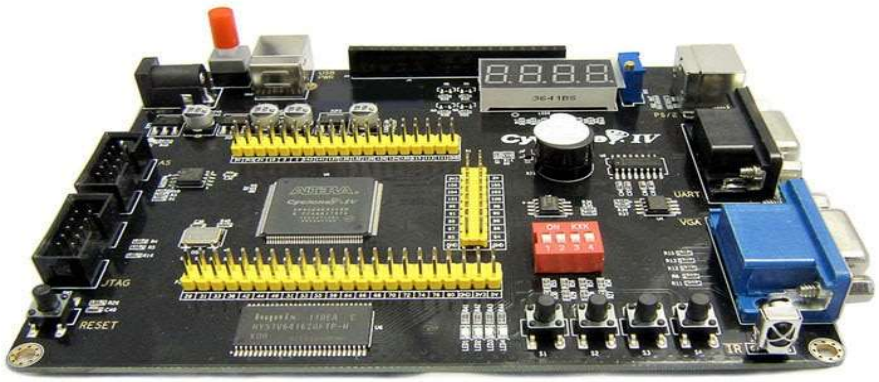


Figura 11. Placa de desarrollo Cyclone IV

3.1.3 Circuitería adicional para las futuras pruebas

Para el ingreso de datos se utilizaron los cinco switches de la placa.

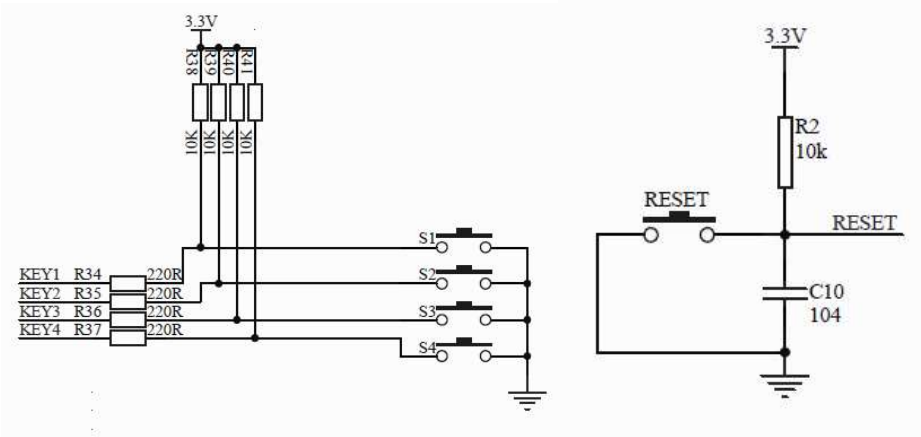


Figura 12. Circuito de swtiches de la placa

Para la visualización de datos se utilizaron 16 leds externos cableados a los GPIOs siguientes:

33	38	42	44	49	51	53	55	64	66	68	70	72	74	76	80
L16	L15	L14	L13	L12	L11	L10	L9	L8	L7	L6	L5	L4	L3	L2	L1

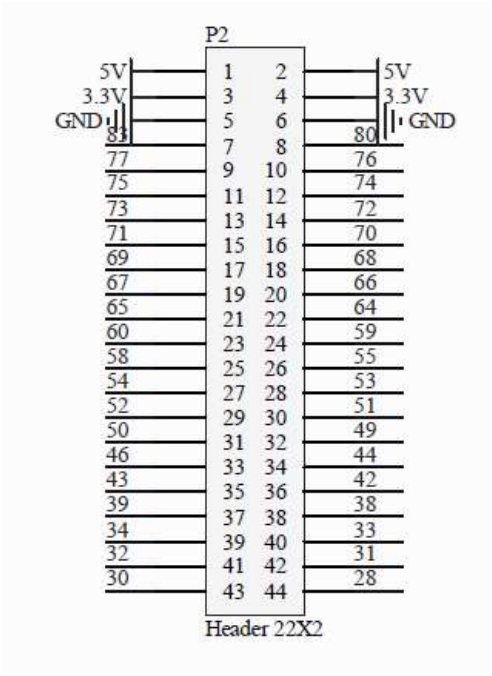


Figura 13. Header 22x2

3.2 Consideraciones previas al diseño de la solución

3.2.1 Formato Q2.30

Los coeficientes del filtro que se han obtenido en la parte de diseño no pueden ser ingresados directamente al FPGA pues representan números en formato decimal y algunos son negativos.

Inicialmente se planteó la opción de utilizar el formato punto flotante como solución. Lamentablemente, en Quartus II no hay un soporte de punto flotante por defecto.

Sin embargo, sí existe un soporte para la representación de un número en formato punto fijo, lo que permitirá la aritmética (suma, resta y multiplicación) de números con signo, sin signo y enteros. Todo ello se realiza mediante el uso de la librería **numeric.std** que viene por defecto con el software Quartus II.

La cantidad de bits a utilizar en la representación digital fue de 32 bits en punto fijo y en complemento a dos. La cantidad de bits para la parte entera fue de 2 y para la parte decimal fue de 30 (Q2.30). Este formato nos permitirá representar números entre -2 y 1.99999.

$$Error\ máximo\ de\ cuantificación = \frac{1}{2^{30}}$$

La solución para este problema fue obtener una nueva forma de representar digitalmente los coeficientes y no perder mucha resolución de estas pues de ello depende la exactitud del filtro a implementar.

3.2.2 Tratamiento de los coeficientes

Una vez planteada la representación de los coeficientes en digital, se procederá a explicar el proceso de acondicionamiento o tratamiento de estos coeficientes para que puedan ser ingresados en el software.

Supongamos que recibimos un dato que ha sido muestreado y el coeficiente siguiente:

$$\text{Muestra: } 18745$$

$$\text{Coeficiente: } 0.007938475$$

$$\text{Producto} = 149$$

Para hacer este producto digitalmente, el coeficiente debe ser multiplicado por 2^{30}, obteniendo como resultado:

$$0.007938475 * 2^{30} = 8523873$$

Este último valor es el que deberemos ingresar al código VHDL.

Posteriormente, el producto del coeficiente (tratado) y la muestra será el siguiente

$$18745 * 8523873 = 159779999385$$

Si la muestra y el coeficiente tienen 32 bits, el producto tendrá 64 bits como resultado de la multiplicación en binario. Sin embargo, el producto obtenido no es el 149 que esperábamos. Para obtener dicho resultado se debe dividir el último producto entre $\mathbf{2^{30}}$. Digitalmente, esto solo requiere el corrimiento a la derecha o shift left de 30 bits.

$$159779999385 \gg 30 = 149$$

3.2.3 Pipeline

Pipeline es una técnica para implementar simultaneidad a nivel de instrucciones dentro de un solo procesador. En nuestro esquema de filtros FIR para cada coeficiente se requiere un multiplicador. Esto quiere decir que para N coeficientes se requerirán N multiplicadores. A nivel de hardware, esto representa una gran cantidad de consumo en recursos lógicos. Partiendo de ello, se realizará un solo bloque multiplicador que interactuará con todos los coeficientes y los registros que almacenan el dato de entrada y los retardos.

3.3 Diseño de la solución digital

3.3.1 Bloque digital propuesto de un filtro FIR

El bloque digital propuesto para el filtro FIR de tercer orden (4 coeficientes) es el que se muestra en la figura.

El bloque tiene los siguientes pines de entrada y salida:

- Data_in: Este vector recibe las muestras con un ancho de banda de 16 bits.
- Sample_valid_in: Controla el inicio del proceso de para la máquina de estados cuando está a nivel alto. Para un sistema con un ADC, esto representaría que la muestra está lista para ser procesada.
- Clk: Representa el reloj para el proceso.
- A0-3: Aquí se ingresan los coeficientes del filtro FIR en un formato de 32 bits.
- Data_out: Representa la salida del filtro FIR, posee un tamaño de16 bits.
- Sample_valid_out: Señal que indica cuando el sistema ha terminado de procesar y valida lo que sale por data_out.
- Data_ext: Representa el retardo (n-4).

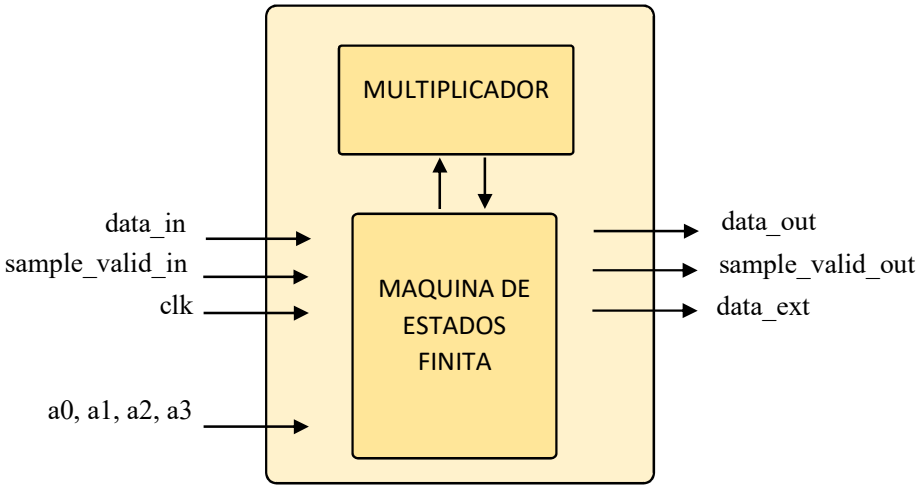


Figura 14. Bloque digital propuesto de un filtro FIR

3.3.2 Bloque digital propuesto de un filtro FIR en cascada

Una vez realizado el bloque fundamental de tercer orden se pasará a hacer la interconexión de varios de estos para así obtener un filtro de mayor de mayor orden.

El bloque digital propuesto tiene las siguientes entradas y salidas.

- Data_in: Este vector recibe las muestras con un ancho de banda de 16 bits.
- syn: Controla el inicio de procesamiento para la muestra de llegada en data_in
- Clk: Representa el reloj para el proceso.
- Data_out: Representa la salida del filtro al finalizar el proceso, el tamaño en bits es de 16.

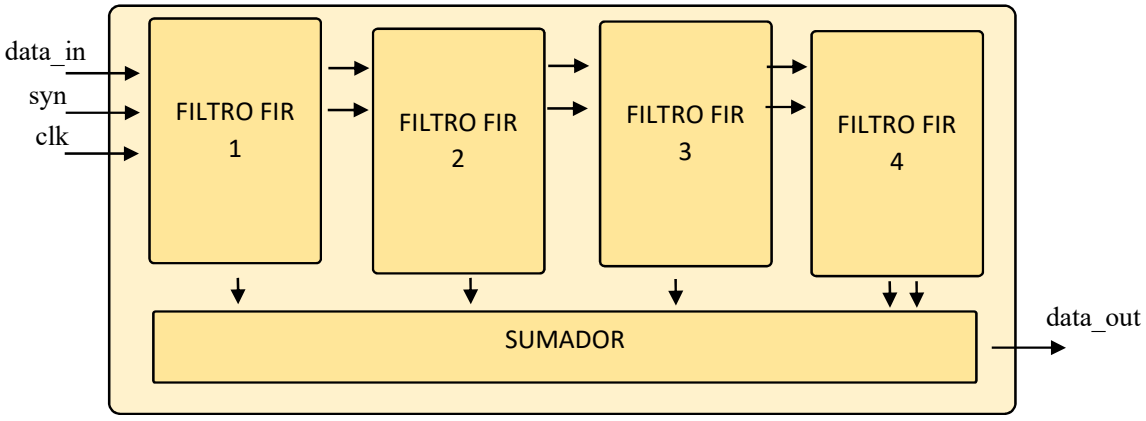


Figura 15. Bloque digital propuesto de un filtro FIR en cascada

3.3.3 Bloque digital propuesto de un filtro polifásico

El mismo bloque fundamental se puede utilizar para la implementación de un filtro polifásico. El objetivo de este tipo de filtros es aumentar la tasa de muestro con la que ingresa la señal de data_in.

El bloque digital propuesto tiene las siguientes entradas y salidas.

- Data_in: Este vector recibe las muestras con un ancho de banda de 16 bits.
- syn: Controla el inicio de procesamiento para la muestra de llegada en data_in
- Clk: Representa el reloj para el proceso.
- Data_out: Representa la salida del filtro al finalizar el proceso, el tamaño en bits es de 16.

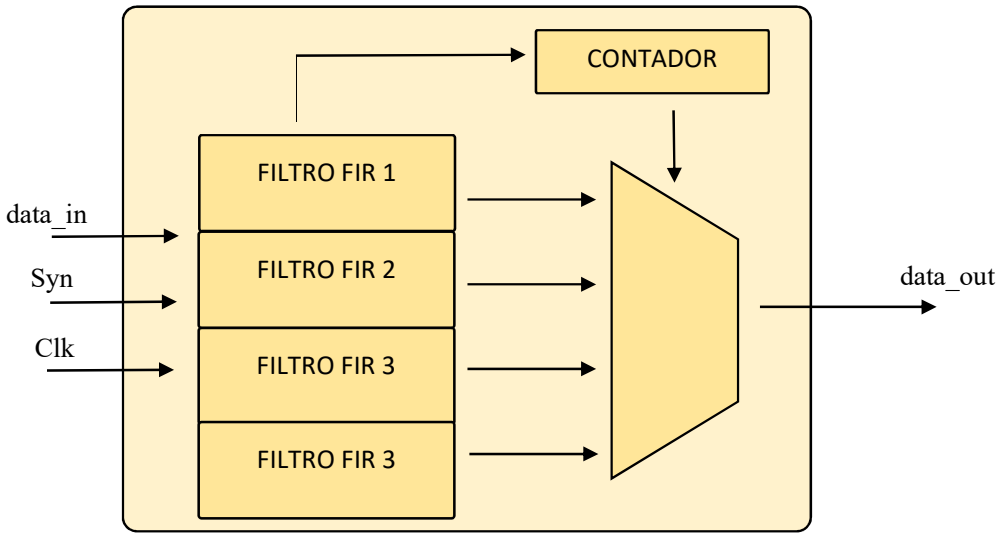


Figura 16. Bloque digital propuesto de un filtro polifásico

3.4 Proceso de diseño e implementación

El proceso de diseño e implementación del filtro se muestra en la figura:

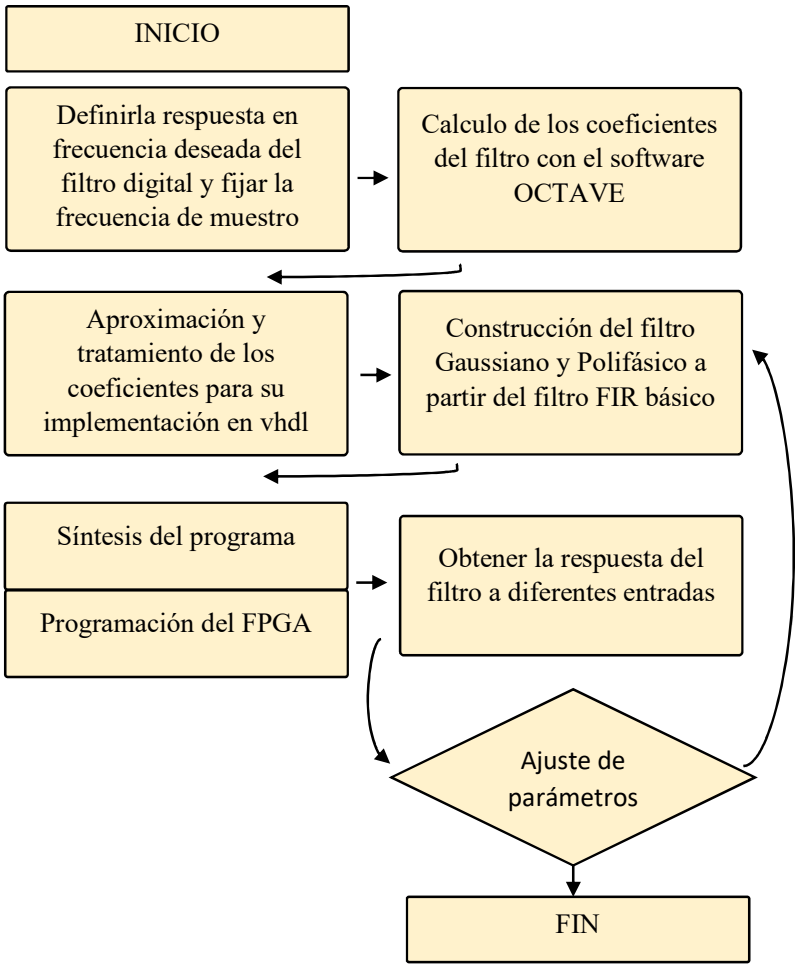


Figura 17. Proceso de diseño e implementación

Capítulo 4 DESARROLLO DE LA SOLUCIÓN

4.1 Equipos y materiales requeridos

El equipo de experimentación necesario para realizar el presente proyecto:

1. PC INTEL CORE i3 o superior
2. Sistema Operativo Windows 10
3. Quartus II Altera
4. Octave V5.2, https://www.gnu.org/software/octave/
5. Matlab R2019

4.2 Desarrollo del filtro gaussiano patrón

Haciendo uso de la función gaussdesign() se obtuvieron los 16 coeficientes de un filtro paso bajo gaussiano.

Coeficientes:

```
h(0) = 0.00158256470120642
h(1) = 0.00460415408995646
h(2) = 0.0116171547697427
h(3) = 0.0254220899213064
h(4) = 0.0482485510670622
h(5) = 0.0794179826231930
h(6) = 0.113374393297969
h(7) = 0.140369467081215
h(8) = 0.150727284896697
h(9) = 0.140369467081215
h(10) = 0.113374393297969
h(11) = 0.0794179826231930
h(12) = 0.0482485510670622
h(13) = 0.0254220899213064
h(14) = 0.0116171547697427
h(15) = 0.00460415408995646
```

Figura 18. Coeficientes del filtro gaussiano

Graficando los coeficientes se observa que generan aproximadamente la curva de gauss:

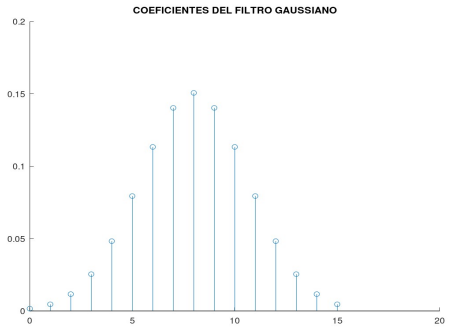


Figura 19. Distribución de coeficientes del filtro gaussiano

Ahora se procederá a realizar un código que nos genere la respuesta al escalón.

```
clear;clc;
#Generamos nuestra señal escalón
#con una amplitud de 1024

t = 35;
n = 0:1:t;
M = 1024*ones(1,36);

#Generamos la matriz de coeficientes

coef = [ 0.00158256470120642        0.00460415408995646
0.0116171547697427        0.0254220899213064
        0.0482485510670622        0.0794179826231930
        0.113374393297969        0.140369467081215
        0.150727284896697        0.140369467081215
        0.113374393297969        0.0794179826231930
        0.0482485510670622        0.0254220899213064
        0.0116171547697427        0.00460415408995646];

#Aplicamos la convolución

filtrada = filter(coef,1,M);

#Graficamos la respuesta
figure(1);
stem(n,filtrada);
title("RESPUESTA ESCALON TEÓRICA");
axis ([0, 35, 0, 1050]);
```

Resultado obtenido

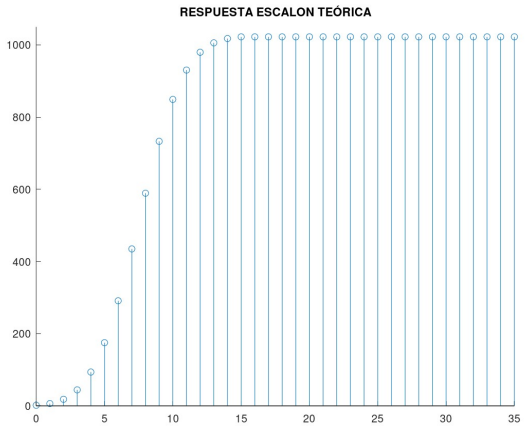


Figura 20. Resultado de la respuesta al escalon del filtro gaussiano

Ahora se procederá a realizar un código que nos genere la respuesta al impulsional.

```
clear;clc;
#Generamos nuestra señal impulso
#con una amplitud de 1024

t = 35;
n = 0:1:t;
M = [1024 zeros(1,35)];

#Generamos la matriz de coeficientes

coef = [ 0.00158256470120642        0.00460415408995646
        0.0116171547697427          0.0254220899213064
        0.0482485510670622          0.0794179826231930
        0.113374393297969           0.140369467081215
        0.150727284896697           0.140369467081215
        0.113374393297969           0.0794179826231930
        0.0482485510670622          0.0254220899213064
        0.0116171547697427          0.00460415408995646];

#Aplicamos la convolución

filtrada = filter(coef,1,M);

#Graficamos la respuesta

figure(1);
stem(n,filtrada);
title("RESPUESTA IMPULSO TEÓRICA");
axis ([0, 35, 0, 200]);
```

Resultado obtenido

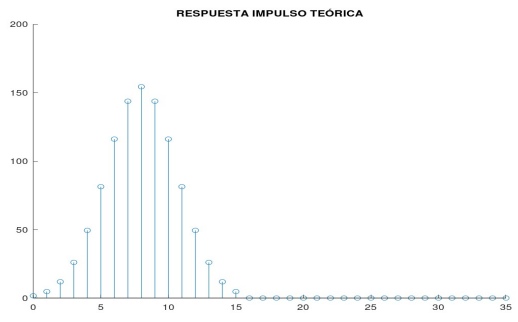


Figura 21. Resultado de la respuesta al impulso
de un filtro gaussiano

4.3 Desarrollo del filtro polifásico patrón

Usando los coeficientes obtenidos en la sección 4.2 se procederá a realizar un código patrón de un filtro polifásico, tanto para una respuesta impulsional como escalón.

Código para un filtro polifásico con entrada escalón

```
clear;clc;
##Generamos nuestro escalón
t = 35;
n = 0:1:t;
M = 1024*ones(1,36);

##Coeficientes filtro gaussiano
b = [0.00158256470120642 0.00460415408995646 0.0116171547697427
0.0254220899213064 0.0482485510670622 0.0794179826231930 0.113374393297969
0.140369467081215 0.150727284896697 0.140369467081215 0.113374393297969
0.0794179826231930 0.0482485510670622 0.0254220899213064 0.0116171547697427
0.00460415408995646];

##Coeficientes de los 4 FILTROS POLIFASICOS
coef1 = [b(1) b(5) b(9) b(13)];
coef2 = [b(2) b(6) b(10) b(14)];
coef3 = [b(3) b(7) b(11) b(15)];
coef4 = [b(4) b(8) b(12) b(16)];

## FILTROS FIR
filtrada1 = filter(coef1,1,M);
filtrada2 = filter(coef2,1,M);
filtrada3 = filter(coef3,1,M);
filtrada4 = filter(coef4,1,M);
N=length(M);
L=4;

##Interpolamos los resultados de cada filtro
y0  = zeros (1, L*N); y0 (1: L: length (y0)) = filtrada1;
y0 = [y0 0 0 0];
y1  = zeros (1, L*N); y1 (1: L: length (y1)) = filtrada2;
y1 = [0 y1 0 0];
y2  = zeros (1, L*N); y2 (1: L: length (y2)) = filtrada3;
y2 = [0 0 y2 0];
y3  = zeros (1, L*N); y3 (1: L: length (y3)) = filtrada4;
y3 = [0 0 0 y3];
n1 = 0:1:(L*N)+L-2;
y = y0 + y1 + y2 + y3;

##Resultados
figure(1)
stem(n1,y);
title("RESPUESTA AL ESCALON TEORICO");
axis([0 25 0 300]);
```

Código para obtener un filtro polifásico con respuesta al impulso

```
clear;clc;
##Generamos nuestro impulso
t = 35;
n = 0:1:t;
M = [1024 zeros(1,35)];

##Coeficientes filtro gaussiano
b = [0.00158256470120642 0.00460415408995646 0.0116171547697427
0.0254220899213064 0.0482485510670622 0.0794179826231930 0.113374393297969
0.140369467081215 0.150727284896697 0.140369467081215 0.113374393297969
0.0794179826231930 0.0482485510670622 0.0254220899213064 0.0116171547697427
0.00460415408995646];

##Coeficientes de los 4 FILTROS POLIFASICOS
coef1 = [b(1) b(5) b(9) b(13)];
coef2 = [b(2) b(6) b(10) b(14)];
coef3 = [b(3) b(7) b(11) b(15)];
coef4 = [b(4) b(8) b(12) b(16)];

## FILTROS FIR
filtrada1 = filter(coef1,1,M);
filtrada2 = filter(coef2,1,M);
filtrada3 = filter(coef3,1,M);
filtrada4 = filter(coef4,1,M);
N=length(M);
L=4;

##Interpolamos los resultados de cada filtro
y0  = zeros (1, L*N); y0 (1: L: length (y0)) = filtrada1;
y0 = [y0 0 0 0];
y1  = zeros (1, L*N); y1 (1: L: length (y1)) = filtrada2;
y1 = [0 y1 0 0];
y2  = zeros (1, L*N); y2 (1: L: length (y2)) = filtrada3;
y2 = [0 0 y2 0];
y3  = zeros (1, L*N); y3 (1: L: length (y3)) = filtrada4;
y3 = [0 0 0 y3];
n1 = 0:1:(L*N)+L-2;
y = y0 + y1 + y2 + y3;

##Resultados
figure(1)
stem(n1,y);
title("RESPUESTA AL IMPULSO TEORICO");
axis([0 25 0 200]);
```

Gráfica obtenida de la respuesta del filtro polifásico a una entrada escalón

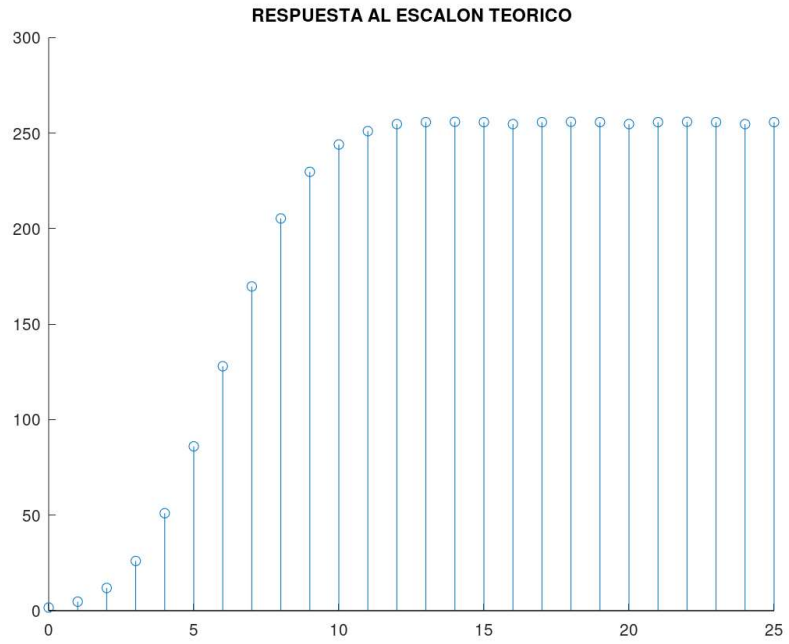


Figura 22. Respuesta al escalon de un filtro polifásico

Gráfica obtenida de la respuesta del filtro polifásico a una entrada impulsional

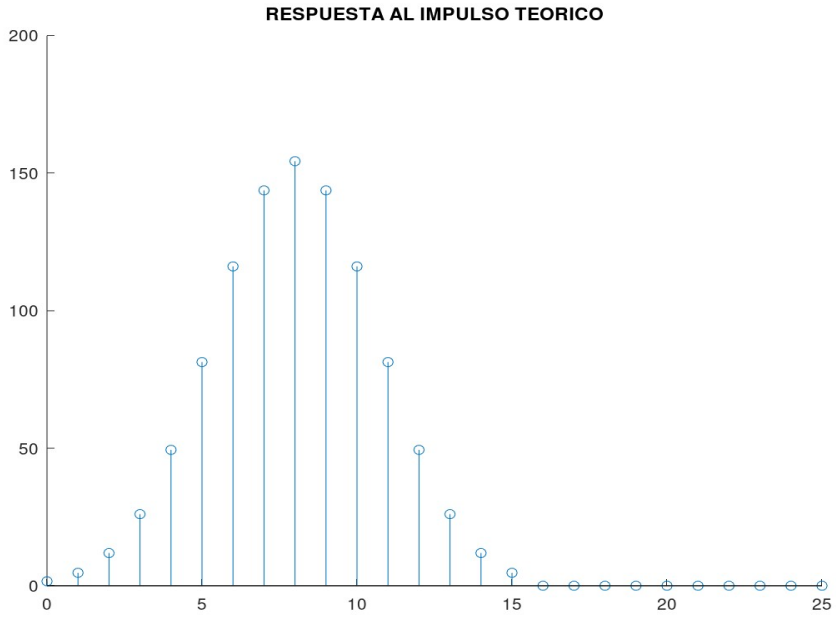


Figura 23. Respuesta al impulso de un filtro polifasico

4.4 Tratamiento previo de los coeficientes

Haciendo uso del software Excel, procederemos a hacer el tratamiento de los coeficientes para ser correctamente ingresados al Quartus II. El proceso consiste en multiplicar la columna de coeficientes por 2^{30} y posteriormente hacer un truncamiento de esta con las funciones que el software Excel nos proporciona.

Tabla 1

COEFICIENTES	POR 2^30	TRUNCAR
0.001582565	1699265.909	1699265
0.004604154	4943672.811	4943672
0.011617155	12473824.95	12473824
0.02542209	27296761.2	27296761
0.048248551	51806487.23	51806487
0.079417983	85274409.52	85274409
0.113374393	121734827.9	121734827
0.140369467	150720567.6	150720567
0.150727285	161842189.8	161842189
0.140369467	150720567.6	150720567
0.113374393	121734827.9	121734827
0.079417983	85274409.52	85274409
0.048248551	51806487.23	51806487
0.02542209	27296761.2	27296761
0.011617155	12473824.95	12473824
0.004604154	4943672.811	4943672

4.5 Creación del bloque FIR de tercer orden

Para hacer nuestro filtro gaussiano y polifásico, se tiene que hacer uso de un bloque FIR de tercer orden fundamental. En la sección 3.3.1 se observa que el bloque consistirá en un arreglo de un multiplicador con una máquina de estados finita. Esta máquina de estados tendrá 6 estados que harán la carga y desplazamiento de los registros (retardos), la asignación de las variables a multiplicar (proceso externo) y la suma total productos almacenados. Se procederá a crear un bloque digital usando el lenguaje VHDL y el software de Quartus II, que realizará la síntesis de este bloque.

Figura 24. Bloque síntesis del filtro
FIR de tercer orden

4.6 Creación del filtro Gaussiano

Haciendo uso del bloque fundamental, se procederá a hacer un filtro gaussiano de 15vo orden. Para dicho fin, se utilizarán 4 de los bloques fundamentales en cascada interconectados mediante el estilo estructural.

Figura 25. Bloque fundamental
del Filtro gaussiano

Haciendo uso de la opción RTL viewer se puede observar las interconexiones del filtro donde se observan los 4 bloques fundamentales en conjunto con la lógica adicional para llevar dicha función.

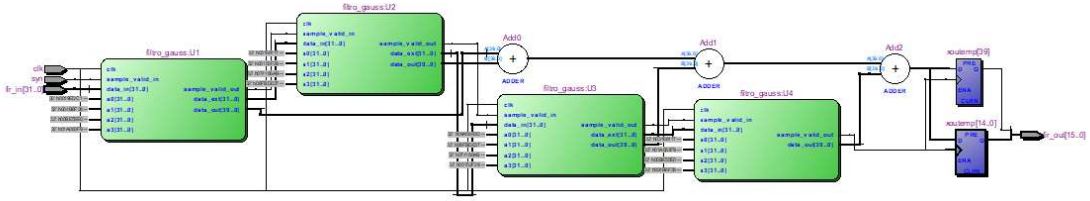


Figura 26. Bloque del filtro gaussiano de 15vo orden

4.7 Creación del filtro polifásico

Haciendo uso del bloque fundamental, se procederá a hacer un filtro polifásico. Para dicho fin, se utilizarán 4 de los bloques fundamentales en paralelo interconectados mediante el estilo estructural.

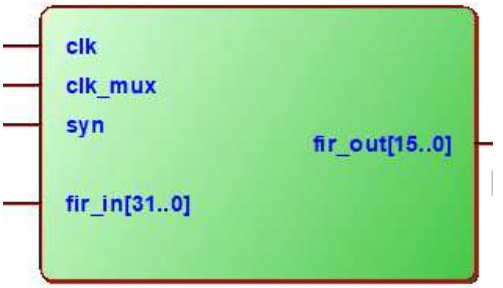


Figura 27. Bloque fundamental
del Filtro gaussiano

Haciendo uso de la opción RTL viewer se puede observar las interconexiones del filtro donde se observan los 4 bloques fundamentales en conjunto con la lógica adicional para llevar dicha función.

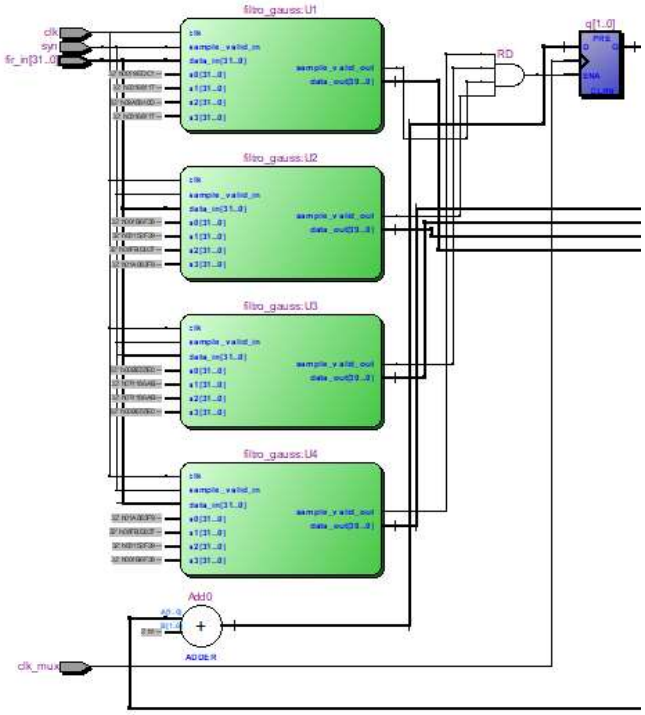


Figura 28. Bloque del filtro polifásico

4.8 Circuito de prueba

Debido al contexto en el que se encuentra el mundo (2021), se hace imposible el uso de los laboratorios de la universidad por lo que se ha propuesto un circuito de prueba manual para verificar que los filtros estén funcionando correctamente.

4.8.1 Filtro Gaussiano

Para verificar el correcto funcionamiento del filtro gaussiano, se hará uso de los cuatro switches del kit de desarrollo del fpga. Con dicho fin, se realizó un circuito adicional que interactuará con el filtro, además de poder visualizar, a través de leds, la salida del filtro.

El bloque digital adicional se basa en bloques anti rebotes para el correcto funcionamiento de los switches y un circuito digital para las entradas del filtro.

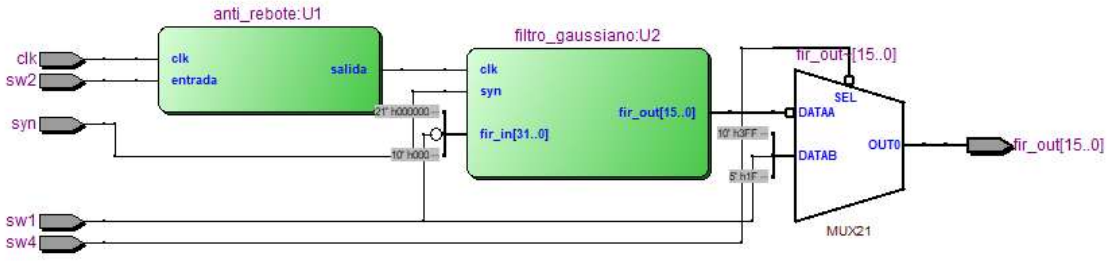


Figura 29. Bloque anti rebotes y filtro gaussiano

ENTRADA	FUNCIÓN
CLK	Reloj del fpga 50MHz
SW2	Representa el reloj del proceso del filtro, físicamente se conecta con el SW2
SYN	Asegura la llegada de un dato, físicamente se conecta con el SW3
SW1	Alterna la entrada del filtro entre 0 y 1024, físicamente está conectado con SW1
SW4	Alterna la visualización de la entrada del filtro con la salida de esta, físicamente está conectado con el SW1
FIR_OUT	Salida a los 16 leds

4.8.2 Filtro Polifásico

Para verificar el correcto funcionamiento del filtro polifásico, se hará uso de los cinco switches del kit de desarrollo del fpga. Con dicho fin, se realizó un circuito adicional que interactuará con el filtro, además de poder visualizar, a través de leds, la salida del filtro.

El bloque digital adicional se basa en bloques anti rebotes para el correcto funcionamiento de los switches y un circuito digital para las entradas del filtro.

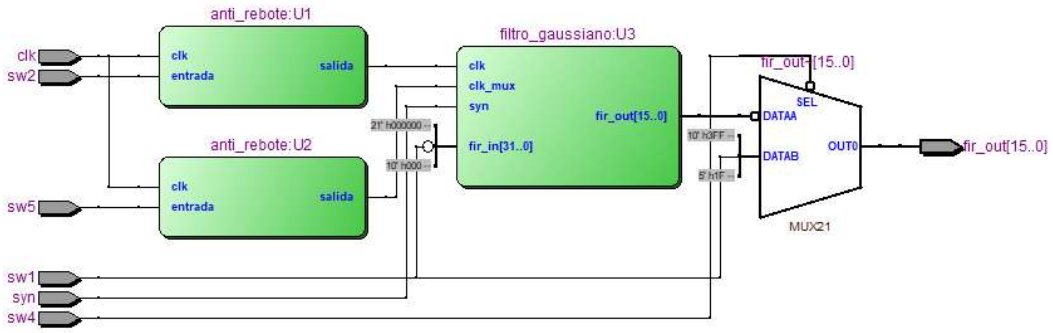


Figura 30. Bloque anti rebotes y filtro polifásico

Tabla 2

ENTRADA	FUNCIÓN
CLK	Reloj del fpga 50MHz
SW2	Representa el reloj del proceso del filtro, físicamente se conecta con el SW2
SYN	Asegura la llegada de un dato, físicamente se conecta con el SW3
SW1	Alterna la entrada del filtro entre 0 y 1024, físicamente está conectado con SW1
SW4	Alterna la visualización de la entrada del filtro con la salida de esta, físicamente está conectado con el SW1
SW5	Representa la señal del multiplexor interno del filtro. Físicamente está conectado al rst del fpga.
FIR_OUT	Salida a los 16 leds

Capítulo 5 PRUEBAS Y RESULTADOS

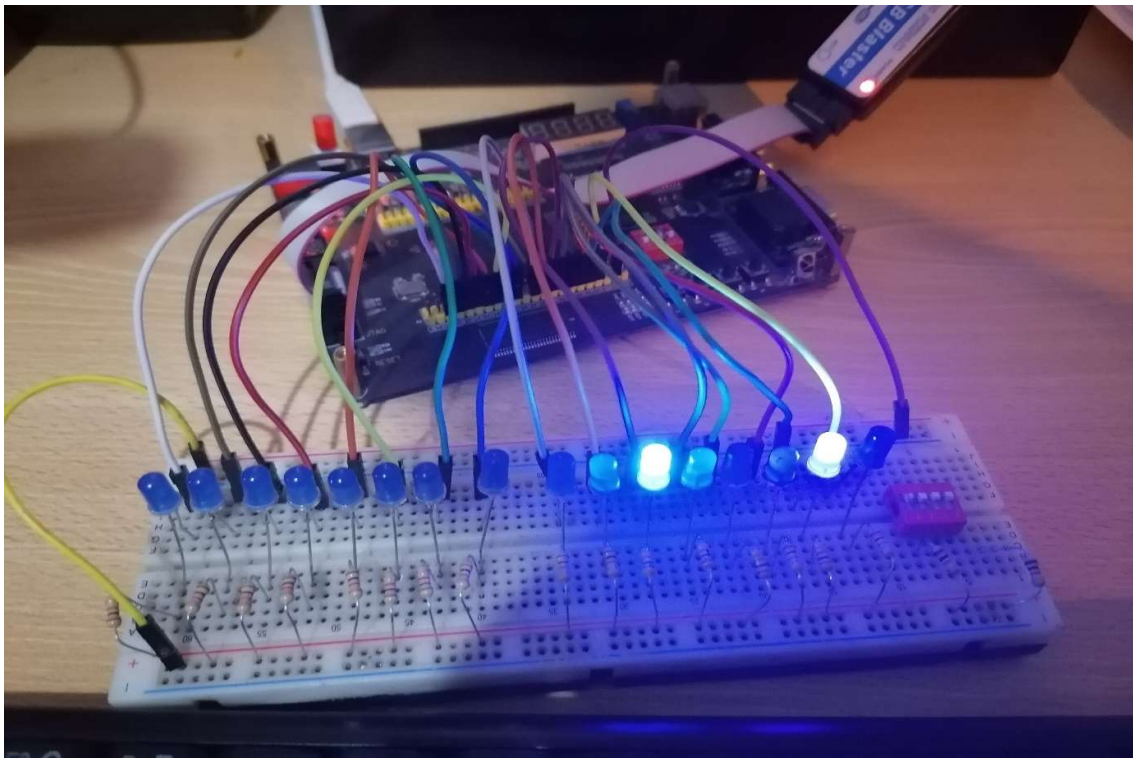


Figura 31. Circuito implementado

Haciendo uso de los switches se ingresaron valores que representaron un impulso y un escalón con una amplitud de 1024. Así mismo, se simuló de forma manual el reloj de proceso y multiplexión de los filtros implementados. Los valores obtenidos se visualizaron en los leds, posteriormente se transformaron a decimales y finalmente se introdujeron a matrices para ser graficados en el software octave. Esto último, nos permitió hacer una comparación práctica con los valores teóricos obtenidos.

5.1 Resultados filtro Gaussiano

Respuesta al escalón

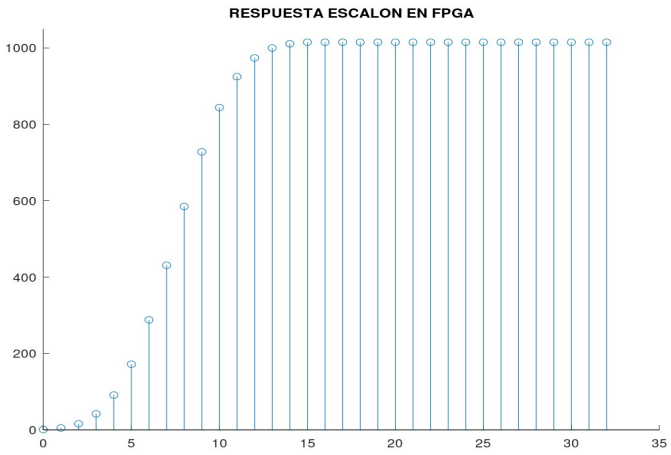


Figura 32. Respuesta al escalon en FPGA del filtro gaussiano

Valores obtenidos vs valores teóricos

Tabla 3

TEORICOS	FPGA
1.6205	1
6.3352	5
18.231	16
44.263	42
93.67	91
174.99	172
291.09	288
434.83	431
589.17	585
732.91	728
849.01	844
930.33	925
979.74	974
1005.8	1000
1017.7	1011
1022.4	1015
1022.4	1015
1022.4	1015
1022.4	1015
1022.4	1015
1022.4	1015
1022.4	1015
1022.4	1015
1022.4	1015

5.2 Resultados filtro Polifásico

Respuesta al impulso

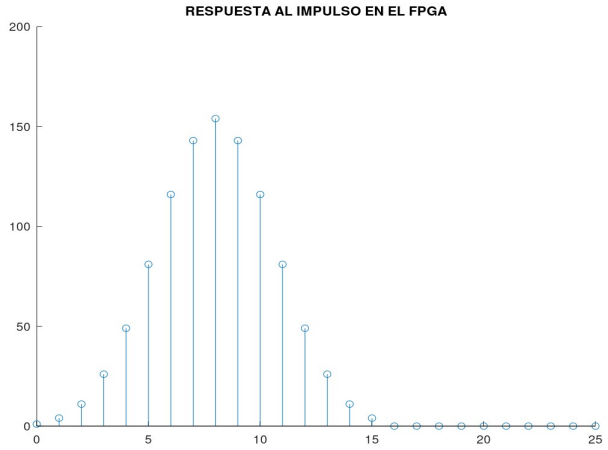


Figura 33. Respuesta al impulso en FPGA del filtro polifasico

Valores obtenidos vs valores teóricos

Tabla 4

TEORICOS	FPGA
1.6205	1
4.7147	4
11.896	11
26.032	26
49.407	49
81.324	81
116.1	116
143.74	143
154.34	154
143.74	143
116.1	116
81.324	81
49.407	49
26.032	26
11.896	11
4.7147	4
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0

Respuesta al escalón

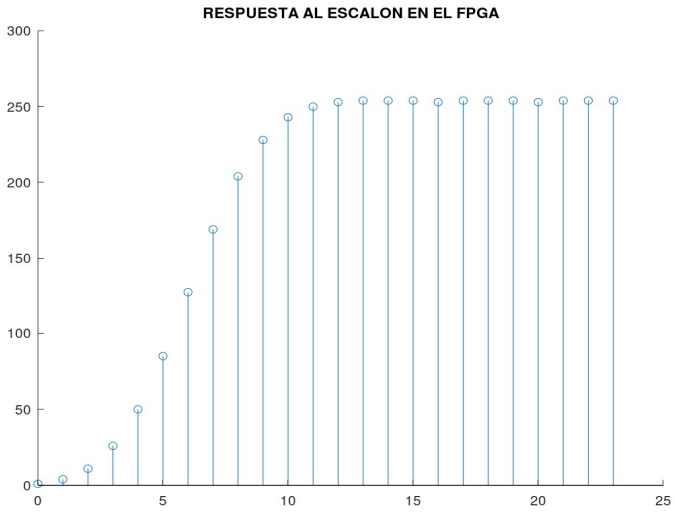


Figura 34. Respuesta al escalon en FPGA del filtro polifasico

Valores obtenidos vs valores teóricos

Tabla 5

TEORICOS	FPGA
1.6205	1
4.7147	4
11.896	11
26.032	26
51.027	50
86.039	85
127.99	127
169.77	169
205.37	204
229.78	228
244.09	243
251.09	250
254.78	253
255.81	254
255.98	254
255.81	254
254.78	253
255.81	254
255.98	254
255.81	254
254.78	253
255.81	254
255.98	254
255.81	254

Capítulo 6 CONCLUSIONES

- Bajo ciertos conocimientos en sistemas digitales y programación, es posible la realización de filtro FIR en un dispositivo FPGA.

- El diseño e implantación de un filtro digital en un FPGA, brinda un mejor entendimiento del hardware que requieren estos sistemas. En base a ello, es posible comparar cuántos recursos conlleva la realización de estos sistemas digitales en equipos comerciales.

- El uso del formato en punto fijo Q2.30 presenta resultados muy cercanos a los teóricos; esto debido a que, al trabajar con un formato entero a la salida del filtro, la resolución de cuantificación se ve levemente afectada.

- Un filtro gaussiano puede realizarse a partir bloques fundamentales de filtro FIR de menor orden, asignándole correctamente los coeficientes.

- Un filtro polifásico se puede realizar a partir de bloques fundamentales de filtro FIR de menor orden, asignándole correctamente los coeficientes.

- La respuesta del filtro gaussiano al escalón es la que esperábamos, esto demuestra que el ancho de banda se verá reducido por lo que será mucho más viable su modulación.

- La respuesta del filtro polifásico fue la esperada, por lo que demuestra que el ancho de banda se verá reducido y la tasa de muestreo se multiplicó por el factor 4 previsto.

- Es posible la verificación de un filtro digital en un FPGA a través de una circuitería adicional a base de LEDS, esto haría posible la implementación rápida en futuros laboratorios para el curso de Procesamiento digital de señales.

Capítulo 7 RECOMENDACIONES

- Para hacer uso de una aritmética binaria efectiva en un dispositivo FPGA se debe hacer uso de la librería numeric.std.

- Para un correcto funcionamiento del sistema en el hardware, se recomienda entender el funcionamiento de los switches.

- Para una mejor performance del filtro, se recomendaría hacer uso de un adc y dac para inyectar señales en tiempo real y hacer un estudio en el tiempo y frecuencia de su salida en un osciloscopio.

- A partir del filtro fundamental, se podrían crear filtros FIR de mayor orden.

- A partir del filtro gaussiano, se pueden implementar sistemas de comunicación avanzadas.

Capítulo 8 BIBLIOGRAFÍA

J. G. Proakis, D. G. Manolakis, “Tratamiento de señales digitales”, 4ta Ed.,Pearson,Madrid, 2007

A. V. Oppenheim, R. W. Schafer, “Discrete-Time Signal Processing”, 2da Ed.,Upper Saddle River, NJ: Prentice-Hall, 1999.

Muñoz Buitron,Dorado Peña(Febrero del 2011), “Diseño de filtro tipo IIR e FIR y adaptativos usando FPGA”, Universidad del Cauca.

Filtro gaussiano “Gaussdesing”, obtenido de mathworks:
https://la.mathworks.com/help/signal/ref/gaussdesign.html

Filtro gaussiano “Gaussian filter Filtro gaussiano ”, obtenido de Wiki:
https://es.qaz.wiki/wiki/Gaussian_filter

Frecuencia de modulación por desplazamiento - Frequency-shift keying , Obtenido de wiki:
https://es.qaz.wiki/wiki/Frequency-shift_keying#Gaussian_frequency-shift_keying

Capítulo 9 ANEXO

ANEXO 1: BLOQUE LOGICO FILTRO FUNDAMENTAL

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity filtro_gauss is
	port(
		clk:				in std_logic := '0';

--Entrada de datos
		data_in:			in signed(31 downto 0) := (others =>'0');
			sample_valid_in:		in std_logic := '0';

--Salida de datos
		data_ext:			out signed(31 downto 0) := (others =>'0');
		data_out:			out signed(39 downto 0) := (others => '0');
		sample_valid_out:		out std_logic := '0';
-- Entrada de datos de los coeficientes
	a0 : integer;
	a1 : integer;
	a2 : integer;
	a3 : integer
					);
end filtro_gauss;

architecture solucion of filtro_gauss is

signal state : integer := 0;

--señales para el multiplicador
signal mult_in_a, mult_in_b :   signed(31 downto 0)  := (others =>'0');
signal mult_out :	signed(63 downto 0) := (others =>'0');
signal temp :		signed (39 downto 0) := (others=>'0');

--señales temporales
signal in_z0, in_z1, in_z2, in_z3 : signed (31 downto 0)  := (others =>'0');


begin

---multiplicador
process(mult_in_a,mult_in_b)
begin
	mult_out <= mult_in_a*mult_in_b;
end process;
```

```
process(clk)
begin
if rising_edge(clk) then

                -- Empeiza el filtro validando que el dato ha llegado
                if (sample_valid_in ='1' and state = 0) then
                                --carga el multiplicador con  data_in * a0
                                data_ext <=in_z3;
                                in_z3 <= in_z2;
                                in_z2 <= in_z1;
                                in_z1 <= in_z0;
                                mult_in_a <=data_in;
                                in_z0 <= data_in;
                                mult_in_b <= to_signed(a0,32);
                                state<= 1;


                elsif (state = 1) then
        --guardamos el producto y la suma temp y aplicamos  un corrimiento de 30
                                temp <= temp + resize(shift_right(mult_out,30),40);
                                mult_in_a <= in_z1;
                                mult_in_b <= to_signed(a1,32);
                                state <= 2;

                elsif (state = 2) then
                        temp <= temp + resize(shift_right(mult_out,30),40);
                        mult_in_a <= in_z2;
                        mult_in_b <= to_signed(a2,32);
                        state <= 3;

                elsif(state = 3) then
                        temp <= temp + resize(shift_right(mult_out,30),40);
                        mult_in_a <= in_z3;
                        mult_in_b <= to_signed(a3,32);
                        state <= 4;

                elsif(state = 4) then
                        temp <= temp+ resize(shift_right(mult_out,30),40);
                        state <= 5;

                elsif(state=5) then
                        data_out <=temp;
                        sample_valid_out <= '1';
                        state <= 6;

                elsif(state = 6) then
                        sample_valid_out <= '0';
                        temp <= (others=>'0');
                        state <= 0;

                end if;
```

```
end if;
end process;
end solucion;
```

ANEXO 2: BLOQUE LOGICO FILTRO GAUSSIANO

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity filtro_gaussiano is
        port(   clk:                    in std_logic;
                syn:                    in std_logic;
                fir_in:                 in std_logic_vector(31 downto 0);
                fir_out:                out std_logic_vector(15 downto 0)
                                        );
end filtro_gaussiano;


architecture solucion of filtro_gaussiano is

signal v1,v2,v3,v4:                             std_logic;
signal fir_in_32bits,temp1,temp2,temp3,temp4:           signed(31 downto 0);
signal xout1,xout2,xout3,xout4,xoutemp:                 signed(39 downto 0);

component filtro_gauss is
        port(           clk:                    in std_logic := '0';
                        --Entrada de datos
                        data_in:                in signed(31 downto 0) := (others =>'0');
                        sample_valid_in:        in std_logic := '0';

--Salida de datos
                        data_ext:               out signed(31 downto 0) := (others =>'0');
                        data_out:               out signed(39 downto 0) := (others => '0');
                        sample_valid_out:       out std_logic := '0';
-- Entrada de datos de los coeficientes
                                a0 : integer;
                                a1 : integer;
                                a2 : integer;
                                a3 : integer
                                );
end component;

begin

fir_in_32bits <= signed(fir_in);

U1: filtro_gauss        port map (
                                clk =>                  clk,
                                data_in =>              fir_in_32bits,
                                sample_valid_in =>      syn,
                                data_ext =>              temp1,
                                data_out =>             xout1,
                                sample_valid_out =>     v1,
```

```
    a0 => 1699265,
    a1 => 4943672,
    a2 => 12473824,
    a3 => 27296761

    );

U2: filtro_gauss        port map (
                                    clk =>              clk,
                                    data_in =>          temp1,
                                    sample_valid_in =>  v1,
                                    data_ext =>         temp2,
                                    data_out =>         xout2,
                                    sample_valid_out =>     v2,

    a0 => 51806487,
    a1 => 85274409,
    a2 => 121734827,
    a3 =>150720567

    );

U3: filtro_gauss        port map (
                                    clk =>              clk,
                                    data_in =>          temp2,
                                    sample_valid_in =>  v2,
                                    data_ext =>         temp3,
                                    data_out =>         xout3,
                                    sample_valid_out =>     v3,

    a0 => 161842189,
    a1 => 150720567,
    a2 => 121734827,
    a3 =>85274409

    );

U4: filtro_gauss        port map (
                                    clk =>              clk,
                                    data_in =>          temp3,
                                    sample_valid_in =>  v3,
                                    data_ext =>         temp4,
                                    data_out =>         xout4,
                                    sample_valid_out =>  v4,

    a0 => 51806487,
    a1 => 27296761,
    a2 => 12473824,
    a3 =>4943672

    );
```

```
process(v4)
begin
                if rising_edge(v4)  then
                                xoutemp  <= xout1+xout2+xout3+xout4;
                end if;
end process;

fir_out <= std_logic_vector(resize(xoutemp,16));

end solucion;
```

ANEXO 3: CIRCUITO DE PRUEBA FILTRO GAUSSIANO

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity top_level is
        port(   clk:                    in std_logic;
                sw1,sw2,syn,sw4:        in std_logic;
--              FIR_IN:                 in std_logic_vector(15 downto 0);
                fir_out:                out std_logic_vector( 15 downto 0)
                                );
end top_level;

architecture solucion of top_level is

signal a,b,rst:                                 std_logic;
signal t,q:                                     std_logic_vector(15 downto 0);
signal fir_in_32bits:                           std_logic_vector(31 downto 0);

component anti_rebote is
port(clk,entrada: in std_logic;
                        salida:         out std_logic);
end component ;

component filtro_gaussiano is
        port(   clk:                            in std_logic;
                syn:                            in std_logic;
                fir_in:                         in std_logic_vector(31 downto 0);
                fir_out:                        out std_logic_vector(15 downto 0)
                                        );
end component;


BEGIN

rst <= '1';
fir_in_32bits <= "0000000000000000"&q;
q <= "0000010000000000" when sw1 = '0'  else (others=>'0');

--process(a)
--begin
--              if rising_edge (a) then
--                      q <=  q + 128;
--              end if;
--end process;
--U0: anti_rebote port map(clk,sw1,a);


U1: anti_rebote port map(clk,sw2,b);
U2: filtro_gaussiano port map(b,syn,fir_in_32bits,t) ;
```

```
fir_out <= not q when sw4 = '0'  else not t;

end solucion;
```

ANEXO 4: BLOQUE LOGICO FILTRO POLIFÁSICO

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity filtro_polifasico is
        port(   clk:                            in std_logic;
                syn:                            in std_logic;
                clk_mux:                        in std_logic;
                fir_in:                         in std_logic_vector(31 downto 0);
                fir_out:                        out std_logic_vector(15 downto 0)
                                                );
end filtro_polifasico;

architecture solucion of filtro_polifasico is

signal v1,v2,v3,v4,RD:                  std_logic;
signal q:                               std_logic_vector(1 downto 0) := (others =>'0');
signal fir_in_32bits,temp1,temp2,temp3,temp4:           signed(31 downto 0);
signal xout1,xout2,xout3,xout4,xoutemp:                 signed(39 downto 0);

component filtro_gauss is
        port(
                        clk:            in std_logic := '0';
--Entrada de datos
                data_in:                        in signed(31 downto 0) := (others =>'0');
                sample_valid_in:                in std_logic := '0';

--Salida de datos
                data_ext:                       out signed(31 downto 0) := (others =>'0');
                data_out:                       out signed(39 downto 0) := (others => '0');
                sample_valid_out:               out std_logic := '0';
-- Entrada de datos de los coeficientes
                                a0 : integer;
                                a1 : integer;
                                a2 : integer;
                                a3 : integer
                                );
end component;

begin

fir_in_32bits <= signed(fir_in);

U1: filtro_gauss        port map (
                        clk =>                  clk,
                        data_in =>              fir_in_32bits,
                        sample_valid_in =>      syn,
                        data_ext =>             temp1,
                        data_out =>             xout1,
```

```
                    sample_valid_out =>     v1,

  a0 => 1699265,
  a1 => 51806487,
  a2 => 161842189,
  a3 => 51806487

  );

U2: filtro_gauss    port map (
                    clk =>              clk,
                    data_in =>          fir_in_32bits,
                    sample_valid_in =>  syn,
                    data_ext =>         temp2,
                    data_out =>         xout2,
                    sample_valid_out =>     v2,

  a0 => 4943672,
  a1 => 85274409,
  a2 => 150720567,
  a3 =>27296761

  );

U3: filtro_gauss    port map (
                    clk =>              clk,
                    data_in =>          fir_in_32bits,
                    sample_valid_in =>  syn,
                    data_ext =>         temp3,
                    data_out =>         xout3,
                    sample_valid_out =>     v3,

  a0 => 12473824,
  a1 => 121734827,
  a2 => 121734827,
  a3 =>12473824

  );

U4: filtro_gauss    port map (
                    clk =>              clk,
                    data_in =>          fir_in_32bits,
                    sample_valid_in =>  syn,
                    data_ext =>         temp4,
                    data_out =>         xout4,
                    sample_valid_out =>     v4,

  a0 => 27296761,
  a1 => 150720567,
  a2 => 85274409,
  a3 =>4943672
```

```
   );

        RD <= v1 and v2 and v3 and v4;

process(clk_mux)
begin
                if RD = '1' then
                        if rising_edge (clk_mux) then
                                q <=  std_logic_vector( unsigned(q) + 1 );
                        end if;
                end if;
end process;

with q select xoutemp <= xout1 when "00",

                        xout2 when "01",

                        xout3 when "10",

                        xout4 when others;


fir_out <= std_logic_vector(resize(xoutemp,16));

end solucion;
```

ANEXO 5: CIRCUITO DE PRUEBA FILTRO POLIFÁSICO

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity top_level is
        port(   clk:                            in std_logic;
                sw1,sw2,syn,sw4,sw5:            in std_logic;
--              FIR_IN:                         in std_logic_vector(15 downto 0);
                fir_out:                        out std_logic_vector( 15 downto 0)
                                );
end top_level;

architecture solucion of top_level is

signal a,b,rst:                 std_logic;
signal t,q:                     std_logic_vector(15 downto 0);
signal fir_in_32bits:           std_logic_vector(31 downto 0);

component anti_rebote is
port(                   clk,entrada:            in std_logic;
                        salida:                 out std_logic);
end component ;

component filtro_gaussiano is
        port(   clk:                    in std_logic;
                syn:                    in std_logic;
                clk_mux:                in std_logic;
                fir_in:                 in std_logic_vector(31 downto 0);
                fir_out:                out std_logic_vector(15 downto 0)
                                        );
end component;
BEGIN

rst <= '1';
fir_in_32bits <= "0000000000000000"&q;
q <= "0000010000000000" when sw1 = '0'  else (others=>'0');


U1: anti_rebote port map(clk,sw2,b);
U2: anti_rebote port map(clk,sw5,a);
U3: filtro_gaussiano port map(b,syn,a,fir_in_32bits,t) ;

fir_out <= not q when sw4 = '0'  else not t;

end solucion;
```

ANEXO 6: BLOQUE LOGICO ANTIREBOTE

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity anti_rebote is
port(clk,entrada: in std_logic;
salida: out std_logic);

end anti_rebote;
architecture solucion of anti_rebote is
type estados is (S0,S1,S2,S3);
signal EA: estados;
signal contador: std_logic_vector(20 downto 0);
signal clk_2: std_logic;
begin

process(clk)
begin
if rising_edge(clk) then
contador <= contador+1;
end if;
end process;
clk_2 <= contador(20);
process(clk_2)
begin
if rising_edge(clk_2) then
case EA is
when S0=> salida <= '0';

if entrada='0' then EA <= S0;
else EA <= S1; end if;

when S1=> salida <= '0';

if entrada='0' then EA <= S0;
else EA <= S2; end if;

when S2=> salida <= '1';

if entrada='0' then EA <= S3;
else EA <= S2; end if;

when S3=> salida <= '1';

if entrada='0' then EA <= S0;
else EA <= S2; end if;

end case;
```

```
end if;
end process;
end solucion;
```